And Inverter Graphs
And Inverter Graphs

Can represent any circuit:

\[ \begin{align*} 
\text{gate 1} & : u \\
\text{gate 2} & : v \\
\text{gate 3} & : w \\
\text{gate 4} & : \text{not } u \\
\text{gate 5} & : \text{not } v \\
\end{align*} \]
And Inverter Graphs

Can represent any functional specification:

\[
\text{sum1: } [4][4] \to [4];
\]
\[
\text{sum1 } xs
\]
\[
= \text{sums } @ \text{width } xs
\]
\[
\text{where sums } = [0] \# [ | x \sim y | | x <- xs | | y <- sums | ];
\]
And Inverter Graphs

Can represent any functional specification (1-bit adder):

```plaintext
addbit: [3] \rightarrow [2];
addbit in
    = num
   where {
    x = (in@0 & ~in@1 & ~in@2) | (~in@0 & in@1 & ~in@2) | 
       (~in@0 & ~inp@1 & in@2) | (in@0 & in@1 & in@2); 
    y = ((in@0 & in@1) | (in@0 & in@2) | (in@1 & in@2));
    num = [x y];
};
```
And Inverter Graphs

Can represent any Boolean function
Can be small vs. CNF representation - ex: xor chain
Can be small vs. BDD representation - ex: multiplier
Can be smaller than the circuit they represent
Equivalence Checking

Formally prove that representations of circuit designs and specifications exhibit exactly the same behavior (over time)

Three principle kinds of circuit tests:

**Synchronous:** show that two design specifications are functionally equivalent if they result in exactly the same sequence of output signals for any valid sequence of input signals.

**Microprocessor:** show that the functions specified for the instruction set architecture will result in exactly the same update of the content of memory that a given RTL implementation does.

**System Design Flow:** show that a Transaction Level Model matches its corresponding RTL
Equivalence Checking

Reminder:

**Transaction Level Model:**
High-level approach to modeling digital systems where details of communication among modules are separated from the details of the implementation of functional units or of the communication architecture. Busses or FIFOs are modeled as channels, and are presented to modules using the SystemC language. Transaction requests take place by calling interface functions of these channel models, which encapsulate low-level details of the information exchange. At the transaction level, the emphasis is on the functionality of the data transfers - what data are transferred to what locations. Transfer protocol is less important. Allows experimentation with different bus architectures, etc. without having to recode models that interact with any of the buses.
Equivalence Checking

Reminder:

**SystemC:**

```cpp
#include "systemc.h"

SC_MODULE(add) // module (class) declaration
{
    sc_in<int> a, b; // ports
    sc_out<int> sum;

    void do_add() // process
    {
        sum = a + b;
    }

    SC_CTOR(add) // constructor
    {
        SC_METHOD(do_add); // register do_add to kernel
        sensitive << a << b; // sensitivity list of do_add
    }
};
```
Equivalence Checking

Reminder:

**Register Transfer Level:**
Describes the behavior of a synchronous digital circuit in terms of the flow of signals, or transfer of data, between hardware registers, and the logical operations performed on those signals.

RTL is used in hardware description languages like VHDL.

Example in VHDL for this circuit:

```vhdl
process(clk)
begin
    if rising_edge(clk) then
        Q <= not Q;
    end if;
end process;
```
Equivalence Checking

Reminder:

**Synchronous circuits:**
- Step 1: Design at RTL using Verilog or VHDL
- Step 2: Verify and/or validate the design
- Step 3: Convert the design to a netlist using logic synthesis tool
- Step 4: Optimize the design
- Step 5: Add circuitry for testing
- Step 6: The Problem - all that mucking may have changed functionality

Step 6: A Solution - test - but who says the tests are complete?

Step 6: Better Solution - equivalence check!!
-- full_adder_bl.vhd

-- full_adder Module - VHDL Gate Level Description

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY full_adder IS
    PORT (
        a : IN BIT;
        b : IN BIT;
        ci : IN BIT;
        s : OUT BIT;
        co : OUT BIT
    );
END full_adder;

ARCHITECTURE gate_level OF full_adder IS
    COMPONENT an02d1 PORT (a1, a2: IN BIT; z: OUT BIT); END COMPONENT;
    COMPONENT xo02d1 PORT (a1, a2: IN BIT; z: OUT BIT); END COMPONENT;
    COMPONENT or03d1 PORT (a1, a2, a3: IN BIT; z: OUT BIT); END COMPONENT;
    -- Intermediate nets
    SIGNAL net1, net2, net3, net4 : BIT;
    BEGIN
        -- Sum Output Bit
        U1 : xo02d1 PORT MAP (a, b, net1);
        U2 : xo02d1 PORT MAP (ci, net1, s);
        -- Carry Output Bit
        U3 : an02d1 PORT MAP (a, b, net2);
        U4 : an02d1 PORT MAP (a, ci, net3);
        U5 : an02d1 PORT MAP (b, ci, net4);
        U6 : or03d1 PORT MAP (net2, net3, net4, co);
    END gate_level;
Equivalence Checking

The beginning of the equivalency check - one circuit has been transformed to an AIG.
Vertex $a$ of the circuit has been merged with vertex $b$ of the AIG.
Equivalence Checking

The completed AIG.
Equivalence Checking

Check nodes 5 and 6.
Equivalence Checking

<table>
<thead>
<tr>
<th>inp/out</th>
<th>vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>0 0 0 0 0 1 1 1 1 1</td>
</tr>
<tr>
<td>middle</td>
<td>0 0 1 1 0 0 1 1 1 1</td>
</tr>
<tr>
<td>bottom</td>
<td>0 1 0 1 0 1 0 1 1 1</td>
</tr>
<tr>
<td>5</td>
<td>0 1 0 1 0 1 0 1 0 0</td>
</tr>
<tr>
<td>6</td>
<td>0 1 0 1 0 1 0 1 0 0</td>
</tr>
</tbody>
</table>

Reduce the graph.
Equivalence Checking

Check nodes 3 and 4.

<table>
<thead>
<tr>
<th>inp/out</th>
<th>vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>0 0 0 0 1 1 1 1 1</td>
</tr>
<tr>
<td>middle</td>
<td>0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>bottom</td>
<td>0 1 0 1 0 1 0 1</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>4</td>
<td>0 0 0 0 0 0 1 0</td>
</tr>
</tbody>
</table>
Equivalence Checking

<table>
<thead>
<tr>
<th>inp/out</th>
<th>vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>0 0 0 0 1 1 1 1 1</td>
</tr>
<tr>
<td>middle</td>
<td>0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>bottom</td>
<td>0 1 0 1 0 1 0 1</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>4</td>
<td>0 0 0 0 0 0 1 0</td>
</tr>
</tbody>
</table>

Reduce the graph.
Equivalence Checking

Check nodes 1 and 2.
Equivalence Checking

Reduce the graph.